IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re patent application of:

Inventor: John S. Worley
Serial No. 10/768,306
Filed: January 29, 2004
For: IMMEDIATE VIRTUAL MEMORY

Examiner: Brian R. Peugh
Group Art Unit: 2187
Docket No. 200208206-1
Date: February 8, 2007

APPEAL BRIEF

Mail Stop: Appeal Briefs – Patents
Commissioner of Patents and Trademarks
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

This appeal is from the decision of the Examiner, in an Office Action mailed September 7, 2006, finally rejecting claims 1-31.

REAL PARTY IN INTEREST

The real party in interest is Hewlett-Packard Development Company, LP, a limited partnership established under the laws of the State of Texas and having a principal place of business at 20555 S.H. 249 Houston, TX 77070, U.S.A. (hereinafter "HPDC"). HPDC is a Texas limited partnership and is a wholly-owned affiliate of Hewlett-Packard Company, a Delaware Corporation, headquartered in Palo Alto, CA. The general or managing partner of HPDC is HPQ Holdings, LLC.

RELATED APPEALS AND INTERFERENCES

Applicant's representative has not identified, and does not know of, any other appeals of interferences which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

STATUS OF CLAIMS

Claims 1-31 are pending in the application. Claims 1-31 were finally rejected in the Office Action dated September 7, 2006. Applicants' appeal the final rejection of claims 1-31, which are copied in the attached CLAIMS APPENDIX.

STATUS OF AMENDMENTS

No Amendment After Final is enclosed with this brief. The last Amendment was filed May 30, 2006.

SUMMARY OF CLAIMED SUBJECT MATTER

Overview

As set forth in the Technical Field section of the current application, the currently claimed invention is "a method and system for efficiently providing default values to software without allocating or initializing the memory pages and without occupying space in the memory hierarchy." The method and system of the current claims are referred to as providing "immediate virtual memory" by Applicant. Immediate virtual memory is a new type of virtual memory made possible by features of the Intel Itanium® architecture and other modern processor architectures and by the currently claimed invention. The phrase "immediate virtual memory" was not previously used in operating systems, to Applicant's knowledge, and no mention of "immediate virtual memory" occurs within any of the cited references. The phrase "immediate virtual memory" was specifically selected as a new, previously unused phrase to describe the new type of virtual memory to which the current application and claims are directed. The phrase "immediate virtual memory" is defined, in great detail, in the current application, as discussed below.

Currently available virtual memory systems provided and supported by currently available operating systems are described in the Background of the Invention

section of the current application, beginning on line 16 of page 1, with reference to Figures 1 and 2. In general, virtual memory systems provide much larger memory address spaces to the processes executing on a computer system than the physical memory resource available within the computer system. This is accomplished by translating virtual memory addresses to physical memory addresses and paging data into physical memory from much higher capacity mass-storage devices and out of physical memory to the much higher capacity mass-storage devices as needed to give the illusion of a larger memory address space to processes executing within the computer system. The background section discusses translation look-aside buffers ("TLB"), a set of specialized, high-speed registers that store most recently accessed virtual-page translations, in the paragraph beginning on line 14 of page 2.

As discussed beginning on line 25 of page 3 of the current application:

> When an operating system or application program begins execution, or when additional memory is allocated for operating system or program use, a computer system commonly allocates a large number of virtual pages on behalf of the operating system or application program. In many cases, the operating system or application program expects that newly allocated virtual pages are initialized to a default value. . . . In many currently available computer systems, the newly allocated virtual memory pages are fully instantiated, meaning that, when sufficient physical memory is available, a physical memory page corresponding to the virtual memory page is assigned for the virtual memory page and that the physical memory page is initialized to the default value.

Figure 2 illustrates the above-mentioned allocation of virtual memory pages on behalf of a process. As shown in Figure 2, the virtual memory pages are allocated in memory, TLB entries corresponding to the virtual memory pages are placed into the TLB, and the physical pages corresponding to the virtual memory pages are initialized to have a default value. In the case shown in Figure 2, the default value is "0." As pointed out in the paragraph beginning on line 4 of page 4, "[i]n general, virtual memory pages are either immediately initialized, under program control, as part of the virtual-page allocation process, or selected from a pool of pre-initialized pages that are zeroed or otherwise initialized in a background, operating-system process." As further pointed out in that paragraph, the initialization of virtual-memory pages is computationally expensive and may involve significant time delays. In the paragraph beginning on line 17 of page 4, Applicant points out that because of the computationally expensive and potentially slow initialization of virtual-memory pages in currently available operating systems, manufacturers, designers, and users of computer systems have all recognized the need for systems and methods that efficiently initialize newly

allocated virtual-memory pages.

As pointed out in the current application, beginning on line 16 of page 7, while the value "0" is most commonly considered the default value for newly initialized virtual-memory pages, other initializations may be desirable. Two examples are then provided, the value "1" and initialization to randomly generated numbers. A third example is provided beginning on line 2 of page 8, in which all bytes within a virtual memory page are initialized to the binary value "0010." On line 17 of page 8, the application makes clear that additional default initializations are possible. Additional examples, mentioned beginning on line 4 of page 8, include algorithmically generated numeric patterns, environmental variables, and other such values. Thus, it is abundantly clear in the current application that a large variety of different repeated-single-value and more complex, computed-value initializations are contemplated as default values provided by the currently claimed immediate virtual memory.

Immediate virtual memory, to which the current application and claims are directed, is well summarized in the summary of the invention section of the current application:

> One or more bit flags within each translation indicate whether or not a corresponding virtual memory page is immediate. READ access to immediate virtual memory is satisfied by hardware-supplied or software-supplied values. WRITE access to immediate virtual memory raises an exception to allow an operating system to allocate physical memory for storing values written to the immediate virtual memory by the WRITE access.

To further summarize, immediate virtual memory results in allocation of physical memory only in the case of a WRITE access. If the virtual memory is accessed only for READ operations, then no physical memory need be allocated, since the values returned to the accessing entity as a result of a READ operation are generated either in hardware or by executable software routines. In other words, the default value or values to which an immediate virtual memory page is initialized are not stored in a physical memory page corresponding to a virtual-memory page, but are instead generated, on each access, by hardware or by executable software routines unless a WRITE access is directed to the virtual memory page.

Figure 3 of the current application illustrates a logical mechanism embodied in processor logic that implements immediate virtual memory. In Figure 3, the virtual page number 310 within a virtual address 308 is used to access the TLB entry 302 corresponding to the virtual address. The TLB entry includes an immediate bit flag 304. When the virtual

address is accessed by a process, the processor checks, in step 312 in the control-flow diagram included in Figure 3, whether or not the immediate bit 304 is set. If the immediate bit is not set, then the processor accesses the page through the standard virtual-memory hierarchy, in step 314. However, if the immediate bit is set, as determined in step 312, then, in step 316, the processor determines whether or not the access to the virtual address is a WRITE or READ access. In the case of a READ access, default data is returned, in step 320. As discussed above, the default data is generated either by hardware or by an executable software routine; and is not actually stored within the page-based memory system of the computer. No physical memory needs to have been allocated for the page in order to fully execute a READ access directed to an immediate virtual memory address. By contrast, if the access is a WRITE access, as determined in step 316, then the processor generates an uninstantiated page exception, in step 318, which invokes exception handling by the operating system to allocate and access the page, as discussed in the current application beginning on line 17 of page 6, which eventually leads to the processor accessing the page through the standard virtual-memory hierarchy, in step 314..

To summarize, immediate virtual memory is virtual memory that is physically allocated by a memory system only in the case that the memory is accessed for a WRITE operation. Immediate virtual memory can be successfully and repeatedly accessed for READ operations without physical allocation and initialization of pages within memory, because immediate virtual memory is considered to have been initialized to a default value, and that default value can be generated in hardware or programmatically by an executable software routine, rather than being fetched from initialized pages within the memory system. The term "immediate" can be thought of as describing the process by which values can be returned, in response to a READ operation, directly, or immediately, without accessing a physical memory page.

Independent Claim 1

Independent claim 1 is directed to a method for providing immediate virtual memory (Current Application, lines 25-31 of page 4) within a computer system. Method steps include: (1) allocating a new translation (302 in Figure 3) for a virtual memory page; and (2) setting one or more bit flags (304 in Figure 3) within the translation to indicate that the translation specifies an immediate virtual memory page.

Dependent Claims 2 – 12

Dependent claim 2 is directed to the method of claim 1 wherein the new translation (302 in Figure 3) is a translation look-aside buffer entry (212 in Figure 2) allocated within a translation look-aside buffer (218 in Figure 2). Dependent claim 3 is directed to the method of claim 2 wherein the new translation look-aside buffer entry is a translation look-aside buffer entry (212 in Figure 2) allocated within a virtual hash page table. Dependent claim 4 is directed to the method of claim 1 wherein the new translation (302 in Figure 3) is allocated within a memory-resident operating-system data structure. Dependent claim 5 is directed to the method of claim 1 wherein, when an immediate virtual memory page is accessed by a READ access instruction, a specified value is returned (320 in Figure 3 and Current Application, line 30 of page 6 to line 3 of page 7). Dependent claim 6 is directed to the method of claim 5 wherein the specified value is generated by a processor logic circuit (Current Application, lines 4-7 of page 7). Dependent claim 7 is directed to the method of claim 5 wherein the specified value is obtained from a default-valued processor register (Current Application, lines 8-10 of page 7). Dependent claim 8 is directed to the method of claim 5 wherein the specified value is specified by one of: (1) a software specification within an operating system; or (2) a hardware logic circuit. Dependent claim 9 is directed to the method of claim 5 wherein the specified value is 0 (Current Application, line 14 of page 7 to line 6 of page 8). Dependent claim 10 is directed to the method of claim 5 wherein the specified value is any fixed, non-zero bit pattern of any size (Current Application, line 14 of page 7 to line 6 of page 8). Dependent claim 11 is directed to the method of claim 5 wherein the specified value is a random number obtained by processor logic algorithmically, from electronic noise, or from another physical source (Current Application, line 14 of page 7 to line 6 of page 8). Dependent claim 12 is directed to the method of claim 1 wherein, when an immediate virtual memory page is accessed by a WRITE access instruction, an exception is generated to allow an operating system to allocate and initialize a physical memory page corresponding to the virtual memory page (Current Application, lines 14-23 of page 6).

Independent Claim 13

Independent claim 13 is directed to a computer processor that provides architecture support (Figure 5 and Current Application, lines 3-30 of page 9) for immediate virtual memory (Current Application, lines 25-31 of page 4). The claimed computer processor includes: (1) processor logic (Current Application, lines 24-25 of page 5) for

executing computer instructions and fetching instructions and data from memory; and (2) processor logic for reading one or more control bits within a translation (302 in Figure 3) and determining whether or not a corresponding unit of memory is immediate (312 in Figure 3 and Current Application, lines 9-13 of page 6).

Dependent Claims 14 – 20

Dependent claim 14 is directed to the computer processor of claim 13 further including: (1) processor logic (Current Application, lines 24-25 of page 5) that, upon READ access to memory determined to be immediate, returns a (320 in Figure 3 and Current Application, line 30 of page 6 to line 3 of page 7); and (2) processor logic that, upon WRITE access to immediate memory, generates an immediate memory exception to allow an operating system to allocate and initialize physical memory corresponding to the immediate memory (Current Application, lines 14-23 of page 6). Dependent claim 15 is directed to the computer processor of claim 14 wherein the specified value is generated by a processor logic circuit (Current Application, lines 4-7 of page 7). Dependent claim 16 is directed to the computer processor of claim 14 wherein the specified value is obtained from a specified processor register (Current Application, lines 8-10 of page 7). Dependent claim 17 is directed to the computer processor of claim 14 wherein the specified value is 0 (Current Application, line 14 of page 7 to line 6 of page 8). Dependent claim 18 is directed to the computer processor of claim 14 wherein the specified value is any fixed, non-zero bit pattern of any size (Current Application, line 14 of page 7 to line 6 of page 8). Dependent claim 19 is directed to the computer processor of claim 14 wherein the specified value is a random number obtained by processor logic (Current Application, lines 24-25 of page 5) algorithmically, from electronic noise, or from another physical source (Current Application, line 14 of page 7 to line 6 of page 8). Dependent claim 20 is directed to the computer processor of claim 14 wherein the specified value is specified by one of: (1) a software specification within an operating system; or (2) a hardware logic circuit.

Independent Claim 21

Independent claim 21 is directed to an operating system that allocates an immediate virtual memory page within a computer system. The operating system allocates an immediate virtual memory page by: (1) allocating a new translation (302 in Figure 3) for the virtual memory page; and (2) setting an immediate bit flag (304 in Figure 3) within the

translation to indicate that the corresponding virtual memory page is immediate, with no allocated physical memory.

Dependent Claim 22-31

Dependent claim 22 is directed to the operating system of claim 21 wherein the new translation (302 in Figure 3) is a translation look-aside buffer entry (212 in Figure 2) allocated within a translation look-aside buffer (218 in Figure 2). Dependent claim 23 is directed to the operating system of claim 22 wherein the new translation look-aside buffer entry (212 in Figure 2) is a translation look-aside buffer entry allocated within a virtual hash page table. Dependent claim 24 is directed to the operating system of claim 21 wherein the new translation (302 in Figure 3) is allocated within a memory-resident operating-system data structure. Dependent claim 25 is directed to the operating system of claim 21 wherein, when an immediate virtual memory page is accessed by a READ access instruction, a specified value is returned (320 in Figure 3 and Current Application, line 30 of page 6 to line 3 of page 7). Dependent claim 26 is directed to the operating system of claim 25 wherein the specified value is generated by a processor logic circuit (Current Application, lines 4-7 of page 7). Dependent claim 27 is directed to the operating system of claim 25 wherein the specified value is obtained from a specified processor register (Current Application, lines 8-10 of page 7). Dependent claim 28 is directed to the operating system of claim 25 wherein the specified value is 0 (Current Application, line 14 of page 7 to line 6 of page 8). Dependent claim 29 is directed to the operating system of claim 25 wherein the specified value is -1 in two's complement arithmetic (Current Application, line 14 of page 7 to line 6 of page 8). Dependent claim 30 is directed to the operating system of claim 25 wherein the specified value is a random number obtained by processor logic (Current Application, lines 24-25 of page 5) algorithmically, from electronic noise, or from another physical source (Current Application, line 14 of page 7 to line 6 of page 8). Dependent claim 31 is directed to the operating system of claim 21 wherein, when an immediate virtual memory page is accessed by a WRITE access instruction, an immediate-virtual-memory-page exception is generated to allow the operating system to allocate and initialize a physical memory page corresponding to the virtual memory page (Current Application, lines 14-23 of page 6).

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

1. Rejection of claims 21-31 under 35 U.S.C. § 101.

2. Rejection of claims 1-31 of the current application under 35 U.S.C. § 102(b) or 35 U.S.C. § 103(a) based on one or more of the cited references.

3. Objection to Figure 3.

ARGUMENT

Claims 1-31 are pending in the current application. In an Office Action dated September 7, 2006 ("Office Action"), the Examiner objected to Figure 3, rejected claims 21-31 under 35 U.S.C. § 101, rejected claims 1-2, 4, 13, 21-22, and 24 under 35 U.S.C. § 102(b) as being anticipated by "Inside Windows NT 2d. Edition," David A. Solomon ("Solomon"), rejected claims 5, 8-9, 12, 14, 17, 20, 25, 28-29, and 31 under 35 U.S.C. § 102(b), as being anticipated by Solomon in view of an email exchange obtained at the Internet address <http://sqid-cache.org/mailarchive/sqi-users/199708/0124.html> ("Wemm") provided in accordance with M.P.E.P. § 2131.01(II), rejected claims 2 and 22 under 35 U.S.C. § 102(b) as anticipated by, or, in the alternative, as obvious under 35 U.S.C. § 103(a) over, Solomon, rejected claims 6-7, 15-16, and 26-27 under 35 U.S.C. § 103(a) as being obvious over Solomon in view of Wemm, the latter being provided in accordance with M.P.E.P. § 2131.01(II) and further in view of "Structured Computer Organization 2d. Edition" by Andrew S. Tanenbaum ("Tanenbaum"), rejected claims 10, 18, and 30 under 35 U.S.C. § 103(a) as being obvious over Solomon in view of Wemm and further in view of LeClerg, U.S. Patent No. 6,857,041 B2 ("LeClerg"), rejected claims 11 and 19 under 35 U.S.C. § 103(a) as being obvious over Solomon in view of Wemm and further in view of Liew, U.S. Patent No. 6,665,249 ("Liew"), and rejected claim 29 under 35 U.S.C. § 103(a) as being obvious over Solomon in view of Wemm in further view of Sakakura et al., US Patent No. 5,625,795 ("Sakakura") and http://www.cs.jcu.edu.au/Subjects/cp1200/1996/org/node9.html ("Sloane").

ISSUE 1

1. Rejection of claims 21-31 under 35 U.S.C. § 101.

In the rejection of claims 21-31 under 35 U.S.C. § 101, the Examiner states:

> Claims 21-31 are rejected under 35 U.S.C. § 101 because the claimed invention is directed to non-statutory subject matter. The claims appear to disclose an operating system without a computer-readable medium needed to realize the operating system's functionality. Such a claimed operating system does not define any structural and functional interrelationships between the operating system and other claimed elements of a computer, which permit the operating system's functionality to be realized.

Subsequently, the Examiner states:

> Regarding Applicant's Argument directed towards the 35 U.S.C. 101 rejection of page 17, the Examiner disagrees. An Operating System has been claimed, however a statutory medium upon which the OS resides, and from which it must be executed from, has not been recited.

Applicant's representative has carefully examined 35 U.S.C. § 101, and cannot find in that statute any requirement for a claim directed towards an operating-system feature to recite "a computer-readable medium needed to realize the operating system's functionality." Applicant's representative has also carefully read the "Interim Guidelines for Examination of Patent Applications for Patent Subject Matter Eligibility," and can find no such requirement there. Lacking a citation to a statute, case law, or USPTO Examination Procedures that indicates a requirement for a claim directed towards an operating-system feature to recite "a computer-readable medium needed to realize the operating system's functionality," this rejection is unfounded and improper.

Indeed, the "Interim Guidelines for Examination of Patent Applications for Patent Subject Matter Eligibility" indicates that the basic 35 U.S.C. § 101 requirement for patentability is for a claimed invention to have a useful, real-world application. An operating system that "allocates an immediate virtual memory page within a computer system by allocating a new translation for the virtual memory page" and "setting an immediate bit flag within the translation to indicate that the corresponding virtual memory page is immediate, with no allocated physical memory," addresses the computational and time overhead associated with allocation of virtual-memory pages during initialization of processes, discussed in the Current Application on lines 17-22 of page 4, by deferring page allocation

until a WRITE operation is directed to the virtual memory pages, as discussed in the Current Application on lines 25-32 of page 4 and on line 14 of page 6 to line 3 of page 7. Increasing the efficiency or process initialization directly increases the overall processing efficiency of a modern computer system. The claimed invention easily satisfies the rather broad standards for patentability expressed in 35 U.S.C. § 101 and in the "Interim Guidelines for Examination of Patent Applications for Patent Subject Matter Eligibility."

ISSUE 2

2. Rejection of claims 1-31 of the current application under 35 U.S.C. § 102(b) or 35 U.S.C. § 103(a) based on one or more of the cited references.

The phrase "immediate virtual memory" is not currently a term of art, but instead was devised by Applicant to describe a new type of virtual memory that is the subject of the current application, to which the current claims are directed. Immediate virtual memory is concisely summarized on lines 25-31 of the current application:

> Various embodiments of the present invention provide for immediate allocation of virtual memory on behalf of processes running within a computer system. One or more bit flags within each translation indicate whether or not a corresponding virtual memory page is immediate. READ access to immediate virtual memory is satisfied by hardware-supplied or software-supplied values. WRITE access to immediate virtual memory raises an exception to allow an operating system to allocate physical memory for storing values written to the immediate virtual memory by the WRITE access.

In other words, an immediate virtual memory page is represented by a translation with an immediate-virtual-memory bit set to differentiate the immediate virtual memory page from conventional virtual memory pages. No physical page is allocated and initialized for an immediate virtual memory page unless a WRITE operation is directed to the immediate virtual memory page. READ operations directed to the immediate virtual memory page prior to direction of a WRITE operation to the immediate virtual memory page are satisfied, in preferred embodiments of the present invention, by returning values generated by processor logic circuits or read from processor registers, rather than returning values stored in a physical memory page corresponding to a conventional virtual memory page.

None of the cited references teach, mention, or suggest immediate virtual memory, the new type of virtual memory to which the current application and current claims

are directed, and which is clearly and particularly defined in the current application. Instead, the cited references refer to conventional virtual memory implemented by conventional operating systems, as discussed in the Background of the Invention section of the current application. The numerous 35 U.S.C. § 102(b) and 35 U.S.C. § 103(a) rejections offered in the Office Action together comprise a document substantially longer and more complex than the Current Application. Rather than respond to each separate rejection, the cited references are first summarized below, in separate subsections, and an argument based on the fact that none of these references teaches, mentions, or suggests immediate virtual memory follows in a final subsection.

Solomon

Solomon's virtual-memory system does not provide immediate virtual memory. Solomon discusses a very standard, conventional operating system with conventional virtual-memory mechanisms and page-fault handling. For example, in Table 513 on pages 265-266 of Solomon, Solomon lists the various types of page faults handled by the Windows NT operating system:

Table 5-13 Reasons for Access Faults

Reason for Fault	Result
Accessing a page that is not resident in memory but is on disk in a page file or mapped file	Allocate a physical page and read the desired page from disk and into the working set
Accessing a page that is on the standby or modified list	Transition the page to the process or system working set
Accessing a page that has no committed storage (for example, reserved address space or address space that is not allocated)	Access violation
Accessing a page from user mode that can be accessed only in kernel mode	Access violation
Writing to a page that is read-only	Access violation
Accessing a demand-zero page	Add a zero-filled page to the process working set
Writing to a guard page	Guard-page violation (if a reference to a user-mode stack, perform automatic stack expansion)
Writing to a copy-on-write page	Make process-private copy of page and replace original in process or system working set
Referencing a page in system space that is valid but not in the process page directory (for example, if paged pool expanded after the process page directory was created)	Copy page directory entry from master system page directory structure and dismiss exception. (This fault is never pointed to by hardware.)
On a multiprocessor system, writing to a page that is valid but hasn't yet been written to	Set dirty bit in PTE

Inspection of this table reveals that there is no page fault in Windows NT for implementing immediate virtual memory. Windows NT does have a page fault corresponding to accessing a "demand-zero page." In response to that page fault, the operating system adds a zero-filled page to the process working set. This is described, in more detail, on page 266 of Solomon under the bulleted heading "Demand Zero:"

> The desired page must be satisfied with a page of zeroes. The pager looks at the zero page list. If the list is empty, the pager takes a page from the standby list and zeros it. The PTE format is the same as the page file PTE shown above, but the page file number and offset are zeros.

As clearly stated by Solomon, the operating system actually zeros a physical page in handling a "demand-zero" page fault. There is no mention in Solomon of generating default values by hardware or by software-executable routine for return to an accessing process, rather than

allocating and initializing an actual physical memory page. Solomon does not in any way distinguish between READ and WRITE access to demand-zero pages.

On pages 261-262 of Solomon, Solomon describes a standard, conventional translation look-aside buffer. The TLB entries described in Solomon include data portions that contain a physical page number, protection field, valid bit, and usually a dirty bit, indicating the condition of the page to which the cache PTE corresponds. Solomon does not teach, mention, or even suggest an immediate-virtual-memory bit, which is not surprising, since Solomon does not teach, mention, or suggest immediate virtual memory.

Finally, Solomon describes virtual address descriptors on pages 273-275, as follows:

> *The memory manager uses a demand-paging algorithm to know when to load pages into memory, waiting until some thread references an address and incurs a page fault before retrieving the page from disk.* Like copy-on-write, demand paging is a form of lazy evaluation--waiting to perform a task until it is required.
>
> *The memory manager uses lazy evaluation not only to bring pages into memory but also to construct the page tables required to describe new pages.* For example, when a thread commits a large region of virtual memory with *VirtualAlloc*, the memory manager could immediately construct the page tables required to access the entire range of allocated memory. But what if some of that range is never accessed? Creating page tables for the entire range would be a wasted effort. *Instead, the memory manager waits to create a page table until a thread incurs a page fault, and then it creates a page table for that page.* This method significantly improves performance for processes that reserve and/or commit a lot of memory but access it sparsely.
>
> With the lazy-evaluation algorithm, allocating even large blocks of memory is a fast operation. This performance gain is not without its trade-offs, however: when a thread allocates memory, the memory manager must respond with a range of addresses for the thread to use. Because the memory manager doesn't build page tables until the thread actually accesses the memory, it can't look there to determine which virtual addresses are free. To solve this problem, the memory manager maintains another set of data structures to keep track of which virtual addresses have been reserved in the process's address space and which have not. These data structures are known as virtual address descriptors (VADs). For each process, the memory manager maintains a set of VADs that describes the status of the process's address space. VADs are structured as a self-balancing binary tree to make lookups efficient. A diagram of a VAD tree is shown in Figure 5-15 on the following page.

> When a process reserves address space or maps a view of a section, the memory manager creates a VAD to store any information supplied by the allocation request, such as the range of addresses being reserved, whether the range will be shared or private, whether a child process can inherit the contents of the range, and the page protection applied to pages in the range.
>
> When a thread first accesses an address, the memory manager must create a PTE for the page containing the address. To do so, it finds the VAD whose address range contains the access address and uses the information it finds to fill in the PTE. If the address falls outside the range covered by the VAD, the memory manager knows that the thread did not allocate the memory before attempting to use it and therefore generates an access violation. (emphasis added)

These virtual-address descriptors are used to implement lazy evaluation in bringing pages into memory and constructing page tables required to describe the pages. As is commonly done in most currently available operating systems, the memory manager waits to create a page table and allocate a page until the page is actually accessed by a process. As explicitly stated by Solomon, when a thread first accesses a virtual address, the memory manager creates a page-table entry for the page and allocates the page. Solomon does not in this passage, or in any other passage, distinguish between READ and WRITE access. Thus, Solomon explicitly states that, upon any first access, including a first READ access, a page table entry is created, and a physical memory page allocated, for the accessed virtual-memory address. There is nothing in Solomon to suggest physical allocation only for WRITE accesses, and generating default values in hardware or programmatically for READ access. In other words, Solomon does not teach, mention, or suggest immediate virtual memory, and is therefore unrelated to the currently claimed invention.

Wemm

Apparently Wemm was cited for the mention of demand zeroing of pages. Wemm explains demand zeroing as follows:

> The pages are not "really" attached to your process yet, but when you access them for the first time, the page fault causes the page to be connected to the process address base and zeroed - this saves a necessary zeroing of pages that are allocated but never used.

This is entirely and exactly the demand-zeroing of pages described in Solomon, discussed in the previous subsection. Note that, whenever a page is accessed, regardless of whether the

access is a READ access or a WRITE access, a page fault is generated causing the page to be allocated and zeroed. This is not the same as immediate virtual memory, in which READ accesses do not cause page faults in page allocation, but instead elicit the return of hardware-generated or executable-software-routine-generated default values that are returned to the accessing entity. Like Solomon, Wemm does not teach, mention, or suggest immediate virtual memory, and is unrelated to the currently claimed invention.

LeClerg

LeClerg is apparently cited by the Examiner for the proposition that physical memory can be initialized to any suitable value. Applicant's representative agrees that physical memory can be initialized to any suitable value. Initialization of physical memory is well known, and has been for at least 50 years. However, LeClerg does not teach, mention, or suggest immediate virtual memory.

Tanenbaum

Tanenbaum appears to be cited by the Examiner for the proposition that hardware and software are logically equivalent and therefore any operation performed by software can also be built directly into hardware. While theoretically true, Tanenbaum also qualifies his statement as follows:

> The decision to put certain functions in hardware and others in software is based on such factors as cost, speed, reliability, and frequency of expected changes.

In other words, while it is theoretically true that an arbitrarily complex hardware design can provide the same functionality as an arbitrary software program, it would be, for example, commercially and practically impossible to implement a modern, complex, high-level-language compiler, such as a C++ compiler, in hardware. First, the required logic circuitry would be incredibly complex and expensive to design and fabricate. More importantly, compilers are frequently modified and extended, and often contain myriad bugs and logic flaws that are not revealed until the compiler is actually used. For rapidly changing logical entities, software is the only practical implementation medium, because it can be easily modified and enhanced. For these reasons, it is impractical to implement a modern C++ compiler in hardware. However, Applicant's representative does agree, in principle, that

software and hardware are interchangeable. This fact has been known for at least 50 years. Tanenbaum does not teach, mention, or suggest immediate virtual memory.

Liew and Sakakura

Liew and Sakakura are cited by the Examiner, as LeClerg, for the proposition that physical memory may be initialized to have a randomly generated value or value "-1." As with LeClerg, Applicant's representative appreciates that physical memory can be initialized to any value. As with LeClerg, this has been well known for at least 50 years. However, Liew explicitly states that a pseudo-random number generator is used to generate the random data, rather than a true random-number-generating process, such as using signal noise. Thus, Liew does not quite teach that for which it is cited. Neither Liew nor Sakakura teaches, mentions, or suggests immediate virtual memory.

Sloane

Sloane is cited by the Examiner for the proposition that it is obvious to represent the value "-1" as a two's complement number. Since two's complement representation of digital values has been used for at least 50 years in computer hardware, Applicant's representative happily acknowledges that it is quite obvious to represent the value "-1" in two's complement arithmetic. Sloane does not teach, mention, or suggest immediate virtual memory.

Argument

All three independent claims of the current application recite "immediate virtual memory," refer to an immediate-virtual-memory page, and refer to an immediate bit flag in a virtual-memory translation, as shown below with underlining uses as emphasis:

> 1. A method for providing immediate virtual memory within a computer system, the method comprising:
> allocating a new translation for a virtual memory page; and
> setting one or more bit flags within the translation to indicate that the translation specifies an immediate virtual memory page.
>
> 13. A computer processor that provide architecture support for immediate virtual memory, the computer processor comprising:
> processor logic for executing computer instructions and fetching instructions and data from memory; and

processor logic for reading one or more control bits within a translation and determining whether or not a corresponding unit of memory is immediate.

21. An operating system that allocates an immediate virtual memory page within a computer system by:

allocating a new translation for the virtual memory page; and

setting an immediate bit flag within the translation to indicate that the corresponding virtual memory page is immediate, with no allocated physical memory.

Again, as stated above, immediate virtual memory, as clearly defined in the current application, is represented by a translation with an immediate-virtual-memory bit set to differentiate the immediate virtual memory page from conventional virtual memory pages. No physical page is allocated and initialized for an immediate virtual memory page unless a WRITE operation is directed to the immediate virtual memory page. READ operations directed to the immediate virtual memory page prior to direction of a WRITE operation to the immediate virtual memory page are satisfied, in preferred embodiments of the present invention, by returning values generated by processor logic circuits or read from processor registers, rather than returning values stored in a physical memory page corresponding to a conventional virtual memory page.

The first anticipation rejection of claim 1 based on the Solomon reference is exemplary of the many anticipation and obviousness-type rejections included in the Office Action. The Examiner states:

A method for providing immediate virtual memory within a computer system **(Pg. 219, ¶s 1-2; it is noted that the reserved memory is the immediate virtual memory)**, the method comprising:

Allocating a new translation **(Pgs. 256-259 Page Table Entries (PTE), 273-274 Virtual Address Descriptors (VAD))** for a virtual memory page **(Pgs. 273, ¶4 and 274 ¶The allocation process consists of the allocation of the VAD and the PTE with both structures being part of the translation)**; and

Setting one or more bit flags within the translation to indicate that the translation specifies an immediate virtual page **(Pgs. 258-259 Fig. 5-11, Table 5-12 Valid bit)**.

As discussed above in detail, Solomon's reserved memory relates to a lazy-evaluation method in bringing pages into memory and constructing page tables required to describe the pages. As is commonly done in most currently available operating systems, the memory manager waits to create a page table with virtual-memory translations and waits to allocate a page until the page is actually accessed by a process. As explicitly stated by Solomon, when a thread

first accesses a virtual address, the memory manager creates a page-table entry, i.e. virtual-memory translation, for the page and allocates the page. Solomon does not distinguish between READ and WRITE access. Solomon explicitly states that, upon any first access, including a first READ access, a page table entry is created, and a physical memory page allocated, for the accessed virtual-memory address. There is nothing in Solomon to suggest physical allocation only for WRITE accesses, and generating default values in hardware or programmatically for READ access. In other words, Solomon does not teach, mention, or suggest immediate virtual memory, and is therefore unrelated to the currently claimed invention.

Moreover, the phrase "reserve memory" is explicitly defined by Solomon as follows:

> To solve this problem, the memory manager maintains another set of data structures to keep track of which virtual addresses have been reserved in the process's address space and which have not.

In other words, reserved memory is a range of virtual addresses that have been reserved for future allocation. In Solomon, the phrase "reserved memory" does not refer to virtual-memory translations, which Solomon specifically avoids initially allocating. Instead, the phrase "reserved memory" refers to a range of addresses for which no virtual-memory translations are allocated, and that range of virtual addresses is stored in a set of data structures separate from the virtual-memory page tables and translation.

Thus, the Examiner has chosen to draw a correspondence between the claim language "immediate virtual memory" and a completely unrelated concept mentioned in Solomon, and reject claim 1 based on a mischaracterization of both the phrase "immediate virtual memory" and a mischaracterization of the phrase "reserved memory." Because claim 1 clearly indicates that an immediate virtual-memory page is represented within a computer system by a virtual-memory translation with an immediate bit flag set, and because Solomon's lazy evaluation defers allocation of virtual-memory translations until a virtual-memory page is allocated, Solomon cannot possibly teach, mention, or suggest immediate virtual memory. Moreover, Solomon does not teach, mention, or suggest deferring physical-page allocation until a WRITE operation, and returning values for READ operations, in the interim, by generating the values via a logic circuit or access to the value in a processor register, as immediate virtual memory operates and is clearly defined in the current application.

The valid bit in a conventional virtual-memory management system, described by Solon, is used to indicate whether or not the virtual-memory translation is valid, or, in other words, whether or not the virtual-memory translation can be used to find a corresponding physical page in memory. That is why the bit is called a "valid bit." The valid bit and has nothing at all to do with immediate virtual memory. Again, the Examiner draws a completely unfounded correspondence between a conventional virtual-memory-management-system bit flag and the entirely novel immediate bit flag disclosed in the current application.

Applicant's representative could specifically address each claim rejection, as the first rejection of claim 1 is addressed above, but that would lead to an exceedingly tedious Appeal Brief of great length. In Applicant's representative's opinion, the fact that all independent claims of the current application specifically recite "immediate virtual memory," and also recite "immediate" and other terms and phrases associated with immediate virtual memory, combined with the fact that no cited reference teaches, mentions, or suggests immediate virtual memory, is sufficient to demonstrate that the large number of complexly worded rejections are quite unfounded. Moreover, the anticipation and obviousness-type rejections are all either solely or partly based on Solomon, and, as discussed above, Solomon does not teach, mention, or suggest immediate virtual memory. Because all of the rejections rely on Solomon as teaching immediate virtual memory, and because Solomon does not, all rejections must fail. The current application concisely and accurately explains the concept of immediate virtual memory, and that concept simply cannot be found in the cited references. In addition, neither the phrase "immediate virtual memory," any phrase or term related to virtual memory qualified by the term "immediate," nor any reference to generation of values to return in response to READ operations directed to a virtual-memory page by any means other than reading those values from a corresponding physical memory page can be found in the cited references. The cited references are simply unrelated to immediate virtual memory, as defined in the current application.

ISSUE 3

3. Objection to Figure 3.

The Examiner continues to insist that Figure 3 omits a page-allocation step carried out by an operating system. As explicitly stated in the current application, "Figure 3 shows a logical mechanism embodied in processor logic for implementing uninstantiated virtual

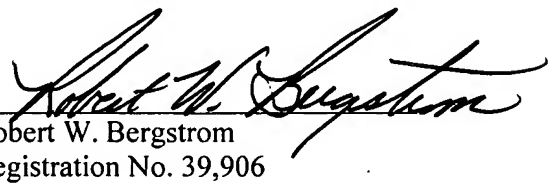
memory pages according to one embodiment." In step 316, "the processor logic determines whether or not the virtual address is being accessed for a WRITE operation." If so, then the processor raises an exception in step 318. Operating systems respond to such exceptions by allocating a page. Processors and processor logic do not. Figure 3 shows processor logic, not an operating system. The Examiner's continued objection is unfounded, and makes no sense. It is not possible to illustrate both processor logic and a complex operating system function in a single diagram, and there is no requirement in any statue, case law, or USPTO rule that would require Applicants to do so.

CONCLUSION

The current claims are directed to patentable subject matter and are directed to a new a type of virtual memory first disclosed in the current application. The cited references do not teach, mention, or suggest immediate virtual memory, as cannot therefore possibly anticipate or make obvious, alone or in any combination, any of the current claims.

Applicant respectfully submits that all statutory requirements are met and that the present application is allowable over all the references of record. Therefore, Applicant respectfully requests that the present application be passed to issue.

Respectfully submitted,
John S. Worley
OLYMPIC PATENT WORKS PLLC

By 
Robert W. Bergstrom
Registration No. 39,906

Olympic Patent Works PLLC
P.O. Box 4277
Seattle, WA 98104
206.621.1933 telephone
206.621.5302 fax

CLAIMS APPENDIX

1. A method for providing immediate virtual memory within a computer system, the method comprising:

allocating a new translation for a virtual memory page; and

setting one or more bit flags within the translation to indicate that the translation specifies an immediate-virtual memory page.

2. The method of claim 1 wherein the new translation is a translation look-aside buffer entry allocated within a translation look-aside buffer.

3. The method of claim 2 wherein the new translation look-aside buffer entry is a translation look-aside buffer entry allocated within a virtual hash page table.

4. The method of claim 1 wherein the new translation is allocated within a memory-resident operating-system data structure.

5. The method of claim 1 wherein, when immediate virtual memory is accessed by a READ access instruction, a specified value is returned.

6. The method of claim 5 wherein the specified value is generated by a processor logic circuit.

7. The method of claim 5 wherein the specified value is obtained from a default-valued processor register.

8. The method of claim 5 wherein the specified value is specified by one of:

a software specification within an operating system; or

a hardware logic circuit.

9. The method of claim 5 wherein the specified value is 0.

10. The method of claim 5 wherein the specified value is any fixed, non-zero bit pattern

of any size.

11. The method of claim 5 wherein the specified value is a random number obtained by processor logic algorithmically, from electronic noise, or from another physical source.

12. The method of claim 1 wherein, when immediate virtual memory is accessed by a WRITE access instruction, an exception is generated to allow an operating system to allocate and initialize a physical memory page corresponding to the virtual memory page.

13. A computer processor that provide architecture support for immediate virtual memory, the computer processor comprising:

processor logic for executing computer instructions and fetching instructions and data from memory; and

processor logic for reading one or more control bits within a translation and determining whether or not a corresponding unit of memory is immediate.

14. The computer processor of claim 13 further including:

processor logic that, upon READ access to memory determined to be immediate, returns a specified value; and

processor logic that, upon WRITE access to immediate memory, generates an immediate memory exception to allow an operating system to allocate and initialize physical memory corresponding to the immediate memory.

15. The computer processor of claim 14 wherein the specified value is generated by a processor logic circuit.

16. The computer processor of claim 14 wherein the specified value is obtained from a specified processor register.

17. The computer processor of claim 14 wherein the specified value is 0.

18. The computer processor of claim 14 wherein the specified value is any fixed, non-zero bit pattern of any size.

19. The computer processor of claim 14 wherein the specified value is a random number obtained by processor logic algorithmically, from electronic noise, or from another physical source.

20. The method of claim 14 wherein the specified value is specified by one of:
a software specification within an operating system; or
a hardware logic circuit.

21. An operating system that allocates an immediate virtual memory page within a computer system by:
allocating a new translation for the virtual memory page; and
setting an immediate bit flag within the translation to indicate that the corresponding virtual memory page is immediate, with no allocated physical memory.

22. The operating system of claim 21 wherein the new translation is a translation look-aside buffer entry allocated within a translation look-aside buffer.

23. The operating system of claim 22 wherein the new translation look-aside buffer entry is a translation look-aside buffer entry allocated within a virtual hash page table.

24. The operating system of claim 21 wherein the new translation is allocated within a memory-resident operating-system data structure.

25. The operating system of claim 21 wherein, when an immediate virtual memory page is accessed by a READ access instruction, a specified value is returned.

26. The operating system of claim 25 wherein the specified value is generated by a processor logic circuit.

27. The operating system of claim 25 wherein the specified value is obtained from a specified processor register.

28. The operating system of claim 25 wherein the specified value is 0.

29. The operating system of claim 25 wherein the specified value is -1 in two's complement arithmetic.

30. The operating system of claim 25 wherein the specified value is a random number obtained by processor logic algorithmically, from electronic noise, or from another physical source.

31. The operating system of claim 21 wherein, when an immediate virtual memory page is accessed by a WRITE access instruction, an immediate-virtual-memory-page exception is generated to allow the operating system to allocate and initialize a physical memory page corresponding to the virtual memory page.

EVIDENCE APPENDIX

None.

RELATED PROCEEDINGS APPENDIX

None.